В.Н. Касьянов

ЯЗЫК ПРЕДСТАВЛЕНИЯ ГРАФОВ GRAPHML: ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ[1]

ВВЕДЕНИЕ

Современное программирование нельзя представить себе без теоретико-графовых методов и алгоритмов [3]. Широкая применимость графов связана с тем, что они являются очень естественным средством объяснения сложных ситуаций на интуитивном уровне. Эти преимущества представления сложных структур и процессов графами становятся еще более ощутимыми при наличии хороших средств их визуализации [4, 11].

Ясно, что инструменты визуализации информации на основе графовых моделей, подобно всем другим инструментам, имеющим дело со структурированными данными, нуждаются в сохранении и передаче графов и ассоциированных с ними данных.

Поэтому неслучайно в 2000 году наблюдательный комитет симпозиума по рисованию графов (Graph Drawing Steering Committee) организовал рабочее совещание по форматам обмена графовыми данными, состоявшееся в г. Вильямсбурге в рамках 8-го симпозиума по рисованию графов (GD-2000) [9]. Как следствие была сформирована неформальная рабочая группа по выработке основанного на языке XML формата обмена графами GraphML, который, в частности, был бы пригоден для обмена данными между инструментами рисования графов и другими приложениями и, в конечном счете, лег бы в основу стандарта описания графов.

Первый отчет по языку вышел в 2001 году [6]. С тех пор язык был расширен в части поддержки основных типов атрибутов и в части включения информации для использования парсерами (синтаксическими анализаторами) [7, 8]. Ведется работа по включению абстрактной информации для описания топологии графа и шаблонов, с помощью которых эту информацию можно преобразовать в различные графические форматы. Программное

[1] Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (грант РФФИ № 12-07-00091)

обеспечение для поддержки работы с GraphML также находится в стадии разработки.

Благодаря XML-синтаксису GraphML может использоваться в комбинации с другими форматами, основанными на XML. С другой стороны, свой собственный механизм расширения позволяет прикреплять <data> метки со сложным содержимым (возможно, требуемый для исполнения с другими моделями XML содержимого) элементов GraphML. Примерами таких меток со сложным содержимым является так называемый механизм SVG (Scalable Vector Graphics) [15], описывающий появление вершин и дуг в изображении. С другой стороны, GraphML может интегрироваться в другие приложения, например, в SOAP сообщения [16].

Простейший способ прочитать или записать GraphML файлы состоит в использовании программного обеспечения, обрабатывающего графы, которое поддерживает данный формат. GraphML является стандартным входным/выходным форматом для системы visone [5] и для графового редактора yEd компании yWorks [24]. Помимо них имеется целый ряд программных инструментов и библиотек, которые либо импортируют, либо экспортируют (либо одновременно и то, и другое) GraphML, включая Pajek [13], ORA [12] и JUNG [16]. Если нужно реализовать привычный GraphML-ридер, можно просто использовать один из многих доступных XML-парсеров и адаптировать его вручную под свои цели.

Базовые средства, образующие ядро языка GraphML, позволяют достаточно адекватно представлять графовые объекты в большинстве приложений. Базовая графовая модель языка охватывает графы[2], которые могут содержать ребра, дуги, петли, кратные дуги, кратные ребра и пометки (атрибуты) вершин и ребер. Множества пометок могут кодировать, например, различные семантические свойства объектов, представленных в виде данной графовой модели, или различные геометрические свойства элементов графа в заданном его изображении на плоскости.

Данная статья продолжает статью [2], в которой представлены базисные средства GraphML, и посвящена дополнительным возможностям языка.

Статья начинается с рассмотрения средств, связанных с расширением базовой графовой модели языка за счет введения таких дополнительных понятий для графовой топологии, как вложенные графы, гиперграфы и порты. Далее описываются два основных способа расширения языка GraphML: за счет добавления новых атрибутов к GraphML-элементам и

---

[2] Здесь и ниже мы без определения используем стандартные понятия из теории графов (см., например, [1]).

путем расширения содержимого элементов <data>. Статья завершается рассмотрением использования XSLT-механизма для преобразования GraphML-документов.

1. РАСШИРЕНИЯ БАЗОВОЙ ГРАФОВОЙ МОДЕЛИ

Для некоторых конкретных приложений базовая графовая модель (см., например, [2]) может оказаться слишком ограниченной и не позволять адекватно моделировать данные прикладной программы. Поэтому авторы предусмотрели в языке расширения данной базовой графовой модели такими конструкциями, как вложенные графы, гиперребра и порты; эти конструкции рассмотрены ниже. Заметим, что все эти дополнительные GraphML-элементы наряду с базовыми могут быть заданы в качестве областей определения функций данных, т.е. в качестве значений для атрибутов <key> (см. [2]).

1.1. Вложенные графы

Помимо атрибутированных смешанных графов, язык GraphML поддерживает и вложенные графы, т.е., графы в которых вершины иерархически упорядочены. Иерархия выражается через структуру GraphML-документа. Вершина в GraphML-документе может иметь элемент <graph>, который содержит вершины, иерархически вложенные в данную вершину.

На рис. 1 и 2 приводится пример вложенного графа и соответствующий ему GraphML-документ. Отметим, что в изображении графа иерархия выражена с помощью включения одного изображения в другое, т.е. вершина u находится в иерархии ниже вершины v, если графическое представление вершины u целиком расположено внутри графического представления вершины v.

Ребра, соединяющие две вершины, находящиеся во вложенных графах, должны быть объявлены в графе, который является в иерархии предком обоих вершин.

Обратите внимание, что в рассмотренном выше примере именно так и сделано. Объявление ребра между вершиной n3 и вершиной n2 в графе G1 было бы неправильно, а объявление их в графе G0 – правильно.

В случае неоднозначности, когда в иерархии существует нескольких общих предков, рекомендуется, но не требуется размещать объявление ребра в наименьшем общем предке конечных вершин данного ребра.

Язык GraphML включает элемент, названный <locator>, который делает возможным определять содержимое части данного документа в другом файле. Более точно, элементы <graph> и <node> могут содержать элемент <locator>, чей атрибут xlink:href указывает на файл, в котором определено содержимое данного элемента <graph> (соответственно, данного элемента <node>). В частности, если элемент <graph> или <node> содержит <locator>, то этот граф <graph>, соответственно, данная вершина <node>, не содержит других элементов. Например, следующий фрагмент документа, который является модифицированной версией документа на рис. 2,

```
<graph id="G0" edgedefault="undirected">
  <node id="n1">
    <graph id="G1" edgedefault="undirected">
      <locator xlink:href="content_of_G1.graphml"/>
    </graph>
  </node>
...
</graph>
```

сообщает парсеру, что содержимое графа с id="G1" определено в содержимом файла G1.graphml. Аналогичным образом содержимое элементов <node> может отсылать к другому файлу с помощью элементов <locator>.

Предполагается, что приложения, которые не поддерживают вложенность графов, могут просто игнорировать все те вершины, которые не принадлежат графу верхнего уровня, а также игнорировать все те ребра, у которых хотя бы одна конечная вершина не принадлежит графу верхнего уровня.

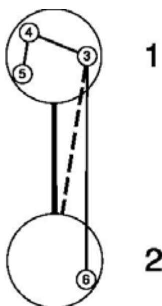

Рис. 1. Вложенный граф

```
<graphml>
  <graph id="G0" edgedefault="undirected">
    <node id="n1">
      <graph id="G1" edgedefault="undirected">
        <node id="n3"/>
        <node id="n4"/>
        <node id="n5"/>
        <edge source="n3" target="n4"/>
        <edge source="n4" target="n5"/>
      </graph>
    </node>
    <node id="n2">
      <graph id="G2" edgedefault="undirected">
        <node id="n6"/>
      </graph>
    </node>
    <edge source="n1" target="n2"/>
    <edge source="n3" target="n2"/>
    <edge source="n3" target="n6"/>
  </graph>
</graphml>
```

Рис. 2. GraphML-документ, описывающий вложенный граф с рис. 1

1.2. Гиперребра

Понятие гиперребра является обобщением понятия обычного ребра в том смысле, что это такое ребро, которое связывает не обязательно только две конечные вершины, но и выражает связь между произвольным числом конечных вершин. Гиперребра объявляются в GraphML-документах с помощью элемента hyperedge. Каждой конечной вершине гиперребра соответствует свой элемент endpoint, входящей в данное гиперребро. Элемент endpoint должен иметь XML-атрибут node, который содержит идентификатор вершины в документе.

Пример гиперграфа и его представления изображены на рис. 3 и 4. Данный гиперграф содержит два гиперребра и два обычных ребра. Гиперребра здесь изображены в виде соединяющихся кривых, а ребра в виде прямых линий.

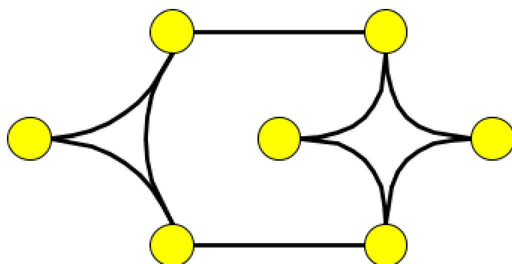

Рис. 3. Пример гиперграфа

Заметим, что ребра можно специфицировать либо элементами <edge>, либо элементами <hyperedge>, содержащими ровно по два элемента <endpoint>. Понятно, что второй способ в большей степени ориентирован на те приложения, которые могут обрабатывать гиперграфы. Элементы <endpoint> имеют факультативный атрибут, называемый типом, который может принимать значения in, out и undir и имеет значение undir по умолчанию.

Предполагается, что приложения, которые не могут обрабатывать гиперребра, будут их просто игнорировать.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5"/>
    <node id="n6"/>
    <hyperedge>
      <endpoint node="n0"/>
      <endpoint node="n1"/>
      <endpoint node="n2"/>
    </hyperedge>
    <hyperedge>
      <endpoint node="n3"/>
```

```
      <endpoint node="n4"/>
      <endpoint node="n5"/>
      <endpoint node="n6"/>
    </hyperedge>
    <hyperedge>
      <endpoint node="n1"/>
      <endpoint node="n3"/>
    </hyperedge>
    <edge source="n0" target="n4"/>
  </graph>
</graphml>
```

Рис. 4. Представление гиперграфа с рис. 3

1.3. Порты

Вершины могут специфицировать различные логические точки для подключения ребер и гиперребер. Такие точки подключения называются *портами*. В качестве аналогии, можно рассматривать граф как материнскую плату, когда вершины представляют интегрированные схемы, а ребра – провода. Тогда контакты интегрированных схем соответствуют портам вершин.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <graph id="G" edgedefault="directed">
    <node id="n0">
      <port name="North"/>
      <port name="South"/>
      <port name="East"/>
      <port name="West"/>
    </node>
    <node id="n1">
      <port name="North"/>
      <port name="South"/>
      <port name="East"/>
      <port name="West"/>
    </node>
    <node id="n2">
      <port name="NorthWest"/>
      <port name="SouthEast"/>
```

```
    </node>
    <node id="n3">
      <port name="NorthEast"/>
      <port name="SouthWest"/>
    </node>
    <edge source="n0" target="n3"
        sourceport="North" targetport="NorthEast"/>
    <hyperedge>
      <endpoint node="n0" port="North"/>
      <endpoint node="n1" port="East"/>
      <endpoint node="n2" port="SouthEast"/>
    </hyperedge>
  </graph>
</graphml>
```

Рис. 5. Представление графа с портами

Порты вершины объявляются с помощью элементов <port>, являющимися детьми по отношению к соответствующему элементу <node>. Порты могут быть вложенными, т.е., они могут содержать внутри себя другие элементы <port>. Каждый элемент <port> должен иметь XML-атрибут name, который является идентификатором данного порта. Элемент <edge> имеет необязательные XML-атрибуты sourceport и targetport, которые задают для ребра начальный и конечный порты, соответственно. Аналогично элемент <endpoint> имеет необязательный XML-атрибут port. Пример GraphML-документа с портами показан на рис. 5.

Предполагается, что те приложения, которые не могут обрабатывать порты, будут их просто игнорировать.

2. РАСШИРЕНИЕ GRAPHML

Язык GraphML спроектирован как легко расширяемый. Базовые средства языка GraphML (см., например, [2]) позволяют описать топологию графа и простые атрибуты его элементов. Для представления более сложных прикладных данных язык GraphML должен быть расширен.

Ниже мы рассмотрим два основных способа расширения языка GraphML: с помощью добавления новых атрибутов к GraphML-элементам (см. разд. 2.1) и путем расширения содержимого элементов <data> за счет разрешения им содержать элементы из других XML-языков (см. разд. 2.2).

Расширения GraphML должны быть заданы в XML-схеме. Схема, в которой определены расширения, может быть порождена из схемы GraphML-документа с помощью стандартного механизма, похожего на механизм, который используется в XHTML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace=http://graphml.graphdrawing.org/xmlns
  xmlns=http://graphml.graphdrawing.org/xmlns
  xmlns:xlink=http://www.w3.org/1999/xlink
  xmlns:xs=http://www.w3.org/2001/XMLSchema
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

<xs:import namespace=http://www.w3.org/1999/xlink
         schemaLocation="xlink.xsd"/>

<xs:redefine
  schemaLocation="http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <xs:attributeGroup name="node.extra.attrib">
    <xs:attributeGroup ref="node.extra.attrib"/>
    <xs:attribute ref="xlink:href" use="optional"/>
  </xs:attributeGroup>
</xs:redefine>

</xs:schema>
```

Рис. 6. Файл graphml+xlink.xsd: определение XML-схемы, которое расширяет GraphML язык путем добавления атрибута xlink:href элементу <node>

2.1. Добавление XML-атрибутов

В большинстве случаев, дополнительная информация может (и должна) быть связана с GraphML-элементами с помощью GraphML-атрибутов (см. [2]), что гарантирует совместимость с другими GraphML-парсерами. Однако в ряде случаев более удобно использовать XML-атрибуты. Предположим, у нас имеется парсер, который умеет обрабатывать XLink-атрибут href и корректно интерпретировать его как URL. Предположим, что нужно сохранить в виде GraphML-документа граф, вершины которого представляют

собой WWW-страницы. Вершина могла бы сослаться на ассоциированную страницу путем сохранения в элементе <node> в качестве значения атрибута xlink:href URL-ссылки на соответствующую страницу:

```
<node id="n0" xlink:href="http://graphml.graphdrawing.org"/>.
```

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns=http://graphml.graphdrawing.org/xmlns
xmlns:xlink=http://www.w3.org/1999/xlink
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
graphml+xlink.xsd">
<graph edgedefault="directed">
<node id="n0" xlink:href="http://graphml.graphdrawing.org"/>
<node id="n1" />
<edge source="n0" target="n1"/>
</graph>
</graphml>
```

Рис. 7. Документ, который можно верифицировать с XSD, показанным на рис. 6. Заметим, атрибут schemaLocation у элемента <graphml> ссылается на файл graphml+xlink.xsd.

Строка

http://graphml.graphdrawing.org

могла бы быть также сохранена в элементе <data>, содержащемся в вершине n0. Однако если сохранять эту строку в качестве значения атрибута xlink:href, то его семантика (т.е. то, что это — URL) становится более очевидной.

Элемент <node>, как он представлен выше, мог стать неверифицируемым в ядре GraphML, поскольку в нем не определен атрибут xlink:href у элемента <node>. Для добавления XML-атрибутов к GraphML-элементам требуется расширить язык GraphML. Это расширение может быть осуществлено с помощью XML-схемы.

Документ на рис. 6 является определением XML-схемы, которая расширяет язык GraphML путем добавления атрибута xlink:href вершине <node>.

Она имеет элемент <schema> в качестве своего корневого элемента (всякое определение XML-схемы обладает этим свойством). Элемент <schema> имеет несколько атрибутов. Строчка

targetNamespace=http://graphml.graphdrawing.org/xmlns

специфицирует, что язык, определенный этим документом, является языком GraphML. Следующие три строчки специфицируют пространство имен по умолчанию (идентифицированное адресом URL для GraphML) и префиксы пространств имен для XLink и XML-схемы. Атрибуты elementFormDefault и attributeFormDefault несущественны в данном примере.

Инструкция

```
<xs:importnamespace=http://www.w3.org/1999/xlink
                    schemaLocation="xlink.xsd"/>
```

предоставляет доступ к пространству имен XLink (предполагается, что определение схемы XLink расположено в файле xlink.xsd).

Само описанное расширение представлено в элементе <redefine>. Атрибут

schemaLocation=http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

у элемента <redefine> специфицирует файл (часть файла), который переопределяется. Фрагмент документа

```
<xs:attributeGroup name="node.extra.attrib">
  <xs:attributeGroup ref="node.extra.attrib"/>
  <xs:attribute ref="xlink:href" use="optional"/>
</xs:attributeGroup>
```

расширяет атрибутную группу, названную node.extra.attrib, которая (по спецификации ядра GraphML) является пустым множеством, но включена в список атрибутов элемента <node>. После переопределения эта атрибутная группа в дополнение к старому содержимому получает один новый атрибут, а именно xlink:href. Этот дополнительный атрибут объявляется в качестве факультативного для элемента <node>.

Аналогично существованию атрибутной группы node.extra.attrib для элемента <node> имеются соответствующие атрибутные группы для всех GraphML-элементов. Эти атрибутные группы пусты в определении ядра GraphML, но могут быть расширены аналогичным образом.

Схема graphml+xlink.xsd может использоваться для валидации документа, показанного на рис. 7.

Авторы языка объясняют свое решение всегда добавлять старое содержимое к вновь определенной атрибутной группе стремлением к тому, чтобы сделать возможным существование более одного определения схемы для расширения одной и той же атрибутной группы.

Запоминание дополнительной информации непосредственно в атрибутах GraphML-элементов, как было проиллюстрировано выше, может показаться более предпочтительным, чем ее сохранение в элементах <data>, как это предусматривают базовые средства языка [2]. По крайней мере, можно заметить, что при таком подходе требуется меньше символов. Однако такое специфицированное пользователем расширение имеет свою цену: поскольку эти нестандартные атрибуты не определятся элементами <key>, GraphML-парсеры будут не в состоянии их обрабатывать.

2.2. Добавление структурного содержимого

В некоторых случаях было бы удобно использовать другие XML-языки для представления данных в GraphML-документах. Например, пользователь может пожелать сохранить изображения вершин, записанные в SVG, как это происходит в следующем фрагменте документа.

```
...
xmlns:svg=http://www.w3.org/2000/svg
...
<node id="n0" >
  <data key="k0">
    <svg:svg width="4cm" height="8cm" version="1.1">
      <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
    </svg:svg>
  </data>
</node>
...
```

Понятно, что атрибуты <svg> и <ellipse> можно также сохранять с помощью функций данных, как это предусматривают базовые средства языка

[2]. Однако представление, приведенное выше, намного более удобно, поскольку приложения, содержащие такое представление, могут использовать существующие парсеры или вьюеры для SVG-изображений.

Язык GraphML можно расширить для верификации документов такого типа. Произвольные элементы могут добавляться к содержимому <data>, но только к элементам <data>, а ядро GraphML не должно быть изменено. Это решение принято для того, чтобы гарантировать способность парсеров всегда понимать структурную часть GraphML-документов и игнорировать возможно неизвестное содержимое элементов <data>.

Рис. 8 содержит определение XML-схемы, которое добавляет SVG-элементы к содержимому <data>.

Схема на рис. 8 подобна примеру на рис. 6. Во-первых, сделаны декларации пространств имен. Затем импортировано пространство имен SVG. Как и прежде, расширение осуществлено в элементе <redefine>. Внутри этого элемента сложный тип data-extension.type расширен SVG-элементом <svg>. Тип data-extension.type является базисным типом для элементов <data> и <default>. Этот тип имеет пустое содержимое в определении ядра GraphML, но может быть расширен произвольными XML-элементами.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://graphml.graphdrawing.org/xmlns"
  xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
>

<xs:import namespace="http://www.w3.org/2000/svg"
           schemaLocation="svg.xsd"/>

<xs:redefine
  schemaLocation="http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <xs:complexType name="data-extension.type">
    <xs:complexContent>
      <xs:extension base="data-extension.type">
        <xs:sequence>
          <xs:element ref="svg:svg"/>
```

```
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:redefine>

</xs:schema>
```

Рис. 8. Файл graphml+svg.xsd: определение XML-схемы, которое добавляет SVG-элемент <svg:svg> к содержимому <data>

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
         xmlns:svg="http://www.w3.org/2000/svg"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
                             graphml+svg.xsd">
 <key id="k0" for="node">
  <default>
   <svg:svg width="5cm" height="4cm" version="1.1">
    <svg:desc>Default graphical representation for nodes
    </svg:desc>
    <svg:rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
   </svg:svg>
  </default>
 </key>
 <key id="k1" for="edge">
  <desc>Graphical representation for edges
  </desc>
 </key>
 <graph edgedefault="directed">
  <node id="n0">
   <data key="k0">
    <svg:svg width="4cm" height="8cm" version="1.1">
     <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
    </svg:svg>
   </data>
  </node>
  <node id="n1" />
```

```
    <edge source="n0" target="n1">
      <data key="k1">
        <svg:svg width="12cm" height="4cm" viewBox="0 0 1200 400">
          <svg:line x1="100" y1="300" x2="300" y2="100"
          stroke-width="5" />
        </svg:svg>
      </data>
    </edge>
  </graph>
</graphml>
```

Рис. 9. Документ, который можно верифицировать с помощью определения XSD, приведенного на рис. 8. Следует заметить, атибут schemaLocation у <graphml> ссылается на graphml+svg.xsd

Таким образом, документы, верифицируемые по схеме с рис. 8, могут иметь элементы <data>, содержащие <svg>. Пример показан на рис. 9. Вершина с id n1 предполагает в качестве графического изображения по умолчанию изображение с ключом k0. Приведенный выше пример демонстрирует полезность пространств имен XML. Здесь имеется два различных элемента <desc>: один в пространстве имен GraphML, а другой в пространстве имен SVG. С помощью различных пространств имен разрешаются конфликты, которые могут возникать из-за элементов, имеющих одинаковые имена в разных XML-языках.

Следует заметить, что имеется не только возможность использовать другие XML-языки (подобно SVG) в GraphML-документах. Сам язык GraphML также можно использовать для представления графовых данных в других расширяемых XML-языках, таких как SVG и XHTML. Данная возможность модульного комбинирования XML-языков гарантирует переиспользуемость парсеров и другого программного обеспечения. Например, SVG-вьюер может вызывать программные системы рисования графов для построения раскладок графов, которые будут сохраняться на языке GraphML внутри SVG-файла.

3. ПРЕОБРАЗОВАНИЕ GRAPHML

Весьма просто обеспечить доступ к изображениям графов на языке GraphML добавлением входных и выходных фильтров для существующего программного обеспечения. Однако разработчики решили, что механизм Extensible Stylesheet Language Transformations (XSLT) [22] предлагает более естественный способ использования XML-данных, в частности, когда результирующий формат некоторого вычисления снова основывается на XML. Отображения, которые переводит входные GraphML-документы в выходные, определяются в виде стилевых XSLT-страниц и могут использоваться отдельно, в качестве компонентов более крупных систем или в веб-сервисах [10].

В основном преобразования определяются в стилевых страницах (style sheets), иногда называемых страницами трансформаций (transformation sheets); в них специфицировано, как входные XML-документы преобразуются в выходные XML-документы в процессе некоторого рекурсивного процесса поиска по образцу. В основе XML-документов лежит модель данных Document Object Model (DOM), некоторое дерево DOM-вершин, изображающее элементы, атрибуты, тексты и т.д., целиком сохраняемые в памяти. Рис. 10 иллюстрирует последовательность шагов XML-преобразования. Вначале XML-данные конвертируются в древовидное представление, которое затем используется для построения результирующего дерева, как это специфицировано в стилевой странице. Наконец, результирующее дерево представляется в виде XML-документа.

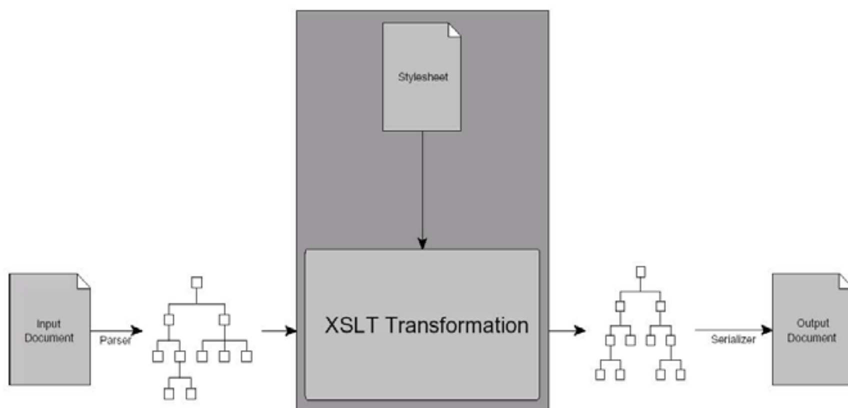

Рис. 10. Последовательность шагов некоторого XSLT-преобразования

По DOM-деревьям можно перемещаться с помощью XPath языка, являющегося подъязыком XSLT. В нем есть средства для представления путей по дереву документа от некоторой конкретной контекстной вершины (подобно дереву директории файловой системы) и нахождения множеств адресов тех его вершин, которые удовлетворяют заданным условиям. Например, если контекстной вершиной является элемент <graph>, все идентификаторы вершин могут адресоваться с помощью child::node/attribute::id, или node/@id в сокращенном виде. Могут использоваться предикаты для более точной спецификации того, какие части DOM-дерева выбираются; например, XPath-выражение edge[@source='n0']/data выбирает только тех <data>-детей ребер <edge>, которые исходят из вершины <node> с заданным идентификатором.

Процесс преобразования грубо можно описать следующим образом. Всякая стилевая страница состоит из некоторого списка шаблонов, каждый из которых имеет некоторый ассоциированный образец и некоторое тело шаблона, описывающее те действия, которые должны быть выполнены, и содержимое, которое должно записано на выходе. Начиная с корня, процессор выполняет поиск в глубину (в порядке записи документа) по DOM-дереву. Для каждой DOM-вершины, куда он попадает, процессор проверяет, есть ли шаблоны с подходящими образцами. Если подходящие шаблоны есть, то выбирается один из них, выполняются действия, содержащиеся в теле данного шаблона (потенциально с дальнейшим поиском по образцу для поддеревьев), и на этом процесс дальнейшего поиска в глубину для DOM-поддеревьев с корнем в данной DOM-вершине прекращается. Если же подходящих шаблонов нет, то процесс поиска в глубину по DOM-дереву продолжается для каждого из сыновей данной DOM-вершины. См. рис. 11 с примером трансформационной XSLT-страницы.

3.1. Средства и типы преобразований

Выразимость и полезность XSLT-преобразований лежит вне их исходной цели добавления некоторого стиля для входа. Ниже приводится обзор важных базисных концепций XSLT и описывается, как эти концепции частично применимы для формулирования продвинутых GraphML-преобразований, которые учитывают помимо структуры DOM-дерева лежащую в его основе комбинаторную структуру графа.

Поскольку язык GraphML проектируется в качестве общего формата, не ограниченного некоторой областью приложений, варианты использования XSLT весьма разнообразны. Однако авторы считают, что все разнообразие

трансформаций можно разделить на три основные категории (типа) в зависимости от реальных целей преобразований. При этом не исключено, что одна и та же трансформация может принадлежать более чем одной из рассмотренных категорий.

Внутренние преобразования. Хотя одной из целей проектирования GraphML является требование хорошо определенной интерпретации всех GraphML-файлов, имеются хорошо известные неоднозначности, связанные с возможностью различных GraphML-представлений для одного и того же графа, например, из-за того, что его вершины <node> и ребра <edge> могут появляться в произвольном порядке. Однако приложения могут требовать, чтобы их GraphML-вход удовлетворял определенным предусловиям, таким, как появление всех вершин <node> перед любым ребром <edge> для того, чтобы строить граф сразу на лету во время его чтения из входного потока.

Среди внутренних преобразований авторы выделяют следующие

- пре- и постпроцессирование GraphML-файла для достижения определенных условий, таких как переупорядочивание выделенных элементов или генерация для них уникальных идентификаторов;
- вставка значений по умолчанию там, где нет явных входов, например, задание ориентации ребер или значений по умолчанию для тэгов <data>,
- разрешение XLink-ссылок в распределенных графах;
- фильтрация ненужных тэгов <data>, которые не релевантны последующей обработке и могут быть опущены с целью сокращения стоимости коммуникации или памяти;
- конвертация между классами графов, например, элиминация гиперребер, подстановка вложенных графов или удаление кратных ребер.

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="iso-8859-1"/>

  <xsl:template match="data|desc|key|default"/> <!--пустой шаблон-->

  <xsl:template match="/graphml">
    <graphml>
      <xsl:copy-of select="key|desc|@*"/>
```

```
      <xsl:apply-templates match="graph"/> <!—обработка graph(s) -->
    </graphml>
  </xsl:template>

  <xsl:template match="graph"> <!—шаблон переоопределения -->
    <graph>
      <xsl:copy-of select="key|desc|@*"/>
      <xsl:copy-of select="node"/> <!--вершины первыми -->
      <xsl:copy-of select="edge"/> <!--затем ребра -->
    </graph>
  </xsl:template>
</xsl:stylesheet>
```

Рис. 11. Пример страницы XSLT-преобразования, удаляющего элементы <data>, <desc>, <key> и <default> из документа и переупорядочивающего вершины и ребра таким образом, чтобы все элементы <node> появились до любого <edge> элемента

Конвертация форматов. Хотя в последнее время GraphML и подобные ему форматы, такие как GXL [23] и GML [15], все шире используются в различных областях, есть еще много приложений и сервисов, которые пока не в состоянии их обрабатывать. Чтобы быть совместимыми, форматы должны конвертироваться один в другой с как можно более полным сохранением информации.

При такой конвертации важно учитывать возможные структурные ошибки в терминах как графовых моделей, так и концепций, которые могут выражаться вовлеченными форматами и их поддержкой дополнительных данных. Конечно, чем ближе концептуальная связь между исходным и целевым форматом, тем , как правило, проще стилевые страницы.

Хотя могут требоваться различные типы таких конвертаций, с точки зрения авторов следует выделить в качестве наиболее важных следующие два случая использования такого типа преобразований.

- Конвертация в другой графовый формат. Ожидается, что GraphML будет использоваться во многих приложениях для архивирования атрибутированных графовых данных и в Web-сервисах для передачи аспектов графа. Хотя легко просто выдавать GraphML, стилевые страницы могут использоваться для конвертации в другие графовые форматы [8] и, таким образом, могут использоваться в транслирующих сервисах, подобных GraphEx [11].

- Экспорт в другие графовые форматы. Конечно, основанные на графах инструменты вообще и инструменты рисования графов в частности должны будут экспортировать в определенные графовые форматы для целей визуализации.

Указанные преобразования необязательно должны применяться файловому документу, они могут также выполняться в памяти приложениями, у которых возникает потребность экспорта в некоторый целевой формат. Следует заметить, что хотя XSLT-преобразования обычно используются для отображения между XML-документами, они могут также использоваться для генерации выходов, которые не являются XML.

Алгоритмические преобразования. Алгоритмические стилевые страницы появляются в преобразованиях, создающих фрагменты выходного документа, которые не соответствуют напрямую фрагментам входного документа, т.е. когда в исходном документе имеется структура, которая не выражается в разметке. Эта ситуация типична для GraphML-данных. Например, нельзя определить, содержит или нет заданный граф <graph> циклы, рассматривая язык разметки; некоторые алгоритмы должны применяться к представленному графу.

Чтобы понять потенциал алгоритмических стилевых страниц, разработчики языка реализовали ряд базовых графовых алгоритмов с использованием XSLT и рекурсивных шаблонов и пришли к выводу, что указанный механизм достаточно мощен для формулирования даже более продвинутых графовых алгоритмов. Например, можно использовать стилевую страницу для вычисления расстояний от единственной входной вершины до всех остальных вершин или реализовывать некоторый алгоритм раскладки, а затем прикрепить результаты к элементам <node>s в виде <data> меток.

3.2. Языковое связывание

Считается, что чистая XSLT-функциональность достаточно выразительна для решения даже более продвинутых связанных с GraphML проблем, чем рассмотренные выше. Однако она имеет ряд общих недостатков, среди которых авторы выделяют следующие.

- С ростом сложности проблем стилевые листы имеют тенденцию становиться все более многословными.
- Алгоритмы необходимо переформулировать в терминах рекурсивных шаблонов, и нет способа использовать уже существующие реализации.
- Вычисления могут плохо выполняться, особенно для больших входов.

Это часто возникает из-за излишнего обхода DOM-дерева и заглядывания вперед, связанных с алгоритмом подстановки шаблонов, встроенным в XSLT-процессор.

- Нет прямого доступа к системным службам, таким как функции данных или связи базовых данных.

Следовательно, большинство XSLT-процессоров разрешают интеграцию функций расширения, реализованных в XSLT или на некоторых других языках программирования. Обычно они по крайней мере поддерживают свой естественный язык. Например, система Saxon [17] может обращаться и использовать внешние Java-классы, поскольку сама она целиком написана на языке Java. В этом случае функции расширения являются методами Java-классов, доступных на пути класса, когда происходит исполнение преобразования и вызов внутри XPath-выражений. Обычно они являются статическими методами, что согласуется с идеей проектирования XSLT в декларативном стиле и без побочных эффектов. Однако XSLT разрешает создавать объекты и вызывать их методы путем связывания созданных объектов с XPath-переменными.

Архитектура, показанная на рис. 12, состоит из следующих трех слоев.

- Стилевая страница, которая создает экземпляр обертки и взаимодействует с ним.
- Класс обертки (реальное XSLT-расширение), которое конвертирует GraphML-разметку в обернутый графовый объект и обеспечивает результаты вычисления.
- Java-классы для графовых структур данных и алгоритмов.

Таким образом, обертка действует как промежуточное звено между графовым объектом и стилевой страницей. Обертка создает экземпляр графового объекта, соответствующего GraphML-описанию, и, например, применяет к нему алгоритм рисования графа. После выполнения он предоставляет результирующие координаты и другие данные по раскладке для того, чтобы стилевая страница вставила в XML (возможно, в GraphML-документ) результат преобразования или продолжила вычисления.

Подход, представленный здесь, является только одним из многих способов отображения файла с внешним описанием графа во внутреннее графовое представление. Отдельное приложение могло бы интегрировать GraphML-парсер, создать свое графовое представление в памяти, отдельное от XSLT, выполнить преобразование и представить результат в виде GraphML-выхода. Однако преимущество использования XSLT состоит в

том, что XSLT генерирует выход естественным способом, и что процесс генерации выхода поддается простому управлению.

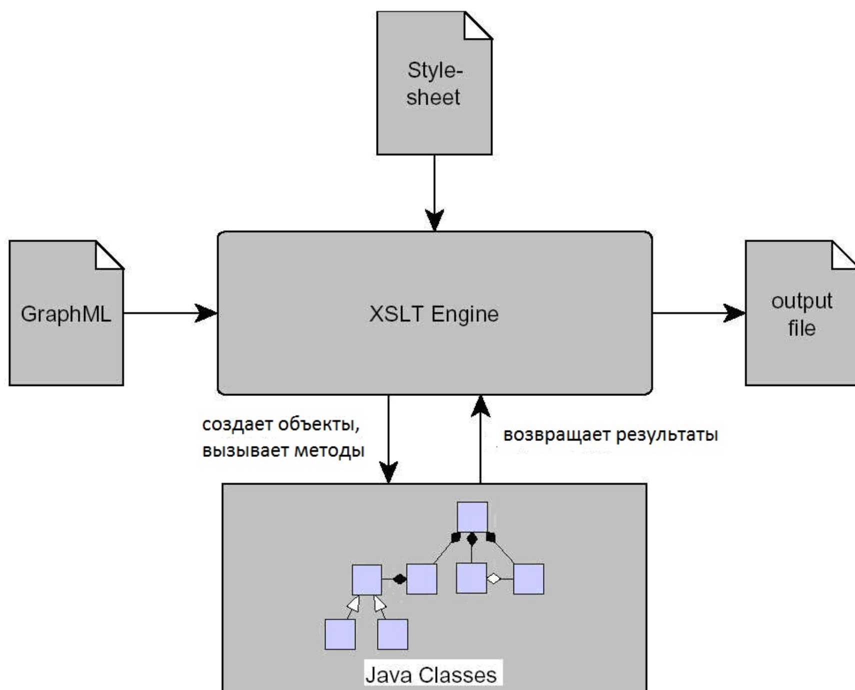

Рис. 12. Использование функций расширения в XSLT

XSL-преобразования образуют простой подход к обработке графов, представленных на языке GraphML. Они доказали свою полезность в различных таких областях применения, где целевой формат некоторого преобразования опять является GraphML-форматом или другим подобным форматом, и когда выходная структура не очень сильно отличается от входной.

Они являются даже достаточно мощными для спецификации преобразований, которые выходят за прямое отображение XML-элементов в другие XML-элементы или другие простые текстовые единицы. Однако такие продвинутые преобразования могут приводить к многословным стилевым страницам, которые сложны для сопровождения и в большинстве представ-

ляются неэффективными. Функции расширения проявили себя в качестве естественного способа борьбы с этими трудностями.

Поэтому механизм XSLT должен использоваться главным образом для выполнения структурных частей преобразования, таких как создание новых элементов или атрибутов, тогда как специализированные расширения более подходят для сложных вычислений, которые при использовании XSLT трудны для выражения или неэффективны для выполнения.

СПИСОК ЛИТЕРАТУРЫ

1. Евстигнеев В. А., Касьянов В. Н. Толковый словарь по теории графов в информатике и программировании. – Новосибирск: Наука, 1999.
2. Касьянов В.Н. Язык представления графов GraphML: базовые средства// Информатика в науке и образовании. – Новосибирск, 2012. – С. 7–22.
3. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003.
4. Касьянов В. Н., Касьянова Е. В. Визуализация графов и графовых моделей. – Новосибирск: Сибирское Научное Издательство, 2010.
5. Baur M., Benkert M., et all. Visone -software for visual social network analysis // Lect. Notes Comput. Sci. – 2002. – Vol. 2265. – P. 463-464. - (Proc. 9th Int. Symp. Graph Drawing GD'2001).
6. Borgatti S.P., Everett M.G., Freeman L.C. UCINET 6.0 // Analytic Technologies, 1999.
7. Brandes U., Eiglsperger M., Herman I., Himsolt M., Marshall M.S. GraphML progress report: structural layer proposal // Proc. 9th Int. Symp. Graph Drawing GD'2001. – Lect. Notes Comput. Sci. – 2002. – Vol. 2265. – P. 501–512.
8. Brandes U., Lerner J., and Pich C. GXL to GraphML and vice versa with XSLT // Electronic Notes in Theoretical Computer Science. – 2004 – Vol. 127, N 1. – P. 113–125.
9. Brandes U., Marshall M.S., and North S.C. Graph data format workshop report // Proc. 8th Int. Symp. Graph Drawing GD'2000. – Lect. Notes Comput. Sci. – 2001. – Vol. 1984. – P. 410–418.
10. Brandes U., Pich C. Graphml transformation // Proc. 11th Int. Symp. Graph Drawing GD'2004. – Lect. Notes Comput. Sci. – 2004. – Vol. 3383. – P. 89–99.
11. Bridgeman S. GraphEx: an improved graph translation service // Proc. 11th Int. Symp. Graph Drawing GD'2004. – Lect. Notes Comput. Sci. – 2004. – Vol. 3383. — P. 307–313.
12. Carley K., Reminga J. ORA: Organization risk analyzer. – Carnegie Mellon University, 2004. – (Tech. Rep. / CMU-ISRI-04-106).
13. De Nooy W., Mrvar A., Batagelj V. Exploratory social network analysis with Pajek. – Cambridge University Press, 2005.

14. Di Battista G., Eades P., Tamassia R., Tollis I.G. Graph Drawing: Algorithms for the Visualization of Graphs. – Prentice Hall, 1999.
15. GML. The Graph Modeling Language File Format. – http://www.infosun.fmi.uni—passau.de/Graphlet/GML/.
16. O'Madadhain J., Fisher D., Smyth P., White S., Boey Y.B. Analysis and visualization of network data using JUNG // Journal of Statistical Software, 2005. – P. 1–35.
17. Saxon Open Source Project. – http://saxon.sourceforge.net/.
18. Sugiyama K., Tagawa S., Toda M. Methods for visual understanding of hierarchical system structures // IEEE Transactions on Systems, Man and Cybernetics. – 1981. – Vol. 11, N 2. — P. 109–125.
19. Tsvetovat M., Reminga J., Carley K. Dynetml: interchange format for rich social network data // NAACSOS Conference. — Pittsburgh, PA, 2003.
20. W3C. Scalable Vector Graphics. – http://www.w3.org/TR/SVG/.
21. W3C. SOAP. – http://www.w3.org/TR/soap12—part0/.
22. W3C. XSL Transformations. – http://www.w3.org/TR/xslt/.
23. Winter A. Exchanging Graphs with GXL // Proc. 9th Int. Symp. Graph Drawing GD'2001. – Lect. Notes Comput. Sci. – 2002. – Vol. 2265. – P. 485–500. – yWork. http://www.yworks.com.